Károlyi Krisztián

Parancsok, szintaktika: (kommunikáció)

Kapcsolódáshoz elég, ha ismerjük a host ip-t és portot. Egy webSocket kapcsolatot kell nyitni. Például python-ben:

```
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((host, port))
```

Ezt követően két külön szálon lehet végtelen ciklusban figyelni a bejövő üzeneteket, vagy küldeni. Pl.

```
message = json.loads(client_socket.recv(1024).decode('utf-8'))
 client_socket.send(message.encode('utf-8'))
```

A JSON példákban természetesen az időpont az mindegy, csak benne hagytam.

Autentikáció

Kliens induláskor a felhasználónevet be kell állítani a további üzenetek küldése előtt. Ezt ugyancsak egy JSON üzenetben küldjük. Egy ilyen üzenet típusa *user_info*, tartalma maga a felhasználónév. A felhasználóvévből a szerver eltávolítja a speciális karakereket, ezeket a kliensen is érdemes lekezelni küldés előtt, hogy ne legyen eltérés. Innen tudja a szerver, hogy a kliens kapcsolat IP címéhez ezt a felhasználónevet kell társítsa. Ez az üzenet valahogy így néz ki például:

```
{
  "message_type": "user_info",
  "content": "krisztian",
  "timestamp": "2023-11-24T12:34:56",
  "sender": " krisztian ",
  "receiver": "server"
}
```

A kapcsolat kialakítása után, az első szerver felé küldött üzenet autentikációs üzenet kell legyen (elküldjük a felhasználónevünket), különben hibaüzenetet küld a szerver és lezárja a kapcsolatunkat! Sikeres azonosítás után kapunk egy üdvözlő üzenetet, amit mindenki megkap:

```
{
  "message_type": "public ",
  "content": "krisztian siekresen csatlakozott, üdv, Krisztián!",
  "timestamp": "2023-11-24T12:35:56",
  "sender": " server ",
  "receiver": "all"
}
```

Publikus üzenetek küldése: a reciver igazából mindegy, de jó, ha „all”-t írunk. Az üzenet típusa *public* kell legyen.

```
{
  "message_type": "public ",
  "content" : ”Sziasztok ",
  "timestamp": "2023-11-24T12:36:52",
  "sender": " krisztian ",
  "receiver": "all"
}
```

Privát üzenet küldése: Valami szintaktikát érdemes kigondolni a kliensben, mindegy, hogy hogy állítjuk össze az inputból az üzenetet, de a küldendő JSON így kell kinézzen:

```
{
  "message_type": "private ",
  "content" : ”Szia, user2 ",
  "timestamp": "2023-11-24T12:39:52",
  "sender": " krisztian ",
  "receiver": "user2"
}
```

erről a szerver küld egy visszajelzést a küldőnek, hogy sikerült-e kézbesíteni-e, vagy sem:

```
{
  "message_type": "private ",
  "content" : ”privát üzenet sikeresen kézbesítve! ",
  "timestamp": "2023-11-24T12:39:55",
  "sender": " server ",
  "receiver": "krisztian"
}
```

vagy

```
{
  "message_type": "private ",
  "content" : ”A privát üzenetet nem sikerült kézbesíteni! user2 nem található. ",
  "timestamp": "2023-11-24T12:39:55",
  "sender": " server ",
  "receiver": "krisztian"
}
```

A fogadónál így érkezik meg:

```
{
  "message_type": "private ",
  "content" : ”Szia, user2 ",
  "timestamp": "2023-11-24T12:39:52",
  "sender": " krisztian ",
  "receiver": "user2"
}
```

Aktív userek lekérése: Igazából mindegy, hogy public, private, stb. típus-e, mert a szerver megnézi, hogy a content @users -e.

```
{
  "message_type": "private ",  //mindegy
  "content" : ”@users",          // ez a lényeg
  "timestamp": "2023-11-24T12:39:52",
  "sender": " krisztian ",
  "receiver": "server"      //mindegy
}
```

erre a válasz egy privát üzenet a szervertől a @users parancs küldőjének: Sajnos szét kell szedni a contentet (darabolni), mivel egy összefűzött stringben érkezik a lista, vesszővel elválasztva a userek. Aktív felhasználók: user1, user2. Emiatt nem lehet vessző a userek nevében.

```
{
  "message_type": "private ",
  "content" : ”Aktív felhasználók: user1, user2",
  "timestamp": "2023-11-24T12:39:52",
  "sender": " server ",
  "receiver": " krisztian"
}
```

új név kérése: @newName név tartalmú üzenettel. Az üzenet típusa itt is mindegy, azt nézi a szerver, hogy ezzel kezdődik-e az üzenet. Ha nem foglalt a név, küld egy broadcast-ot, hogy xy neve megváltozott erre: yx, (frissíti a users listában is, de a db-ben visszamenőleg nem!) különben privát üzenetben visszaküld egy hibaüzenetet, hogy foglalt a név, kérjen újat.

```
{
  "message_type": "private ",   //mindegy
  "content" : ”@newName Kiki ",
  "timestamp": "2023-11-24T12:49:52",
  "sender": " krisztian ",
  "receiver": " server"   //mindegy
}
```

sikeres esetben:

```
{
  "message_type": "public ",
  "content": "krisztian neve megváltozott erre: Kiki",
  "timestamp": "2023-11-24T12:49:54",
  "sender": " server ",
  "receiver": "all"
}
```

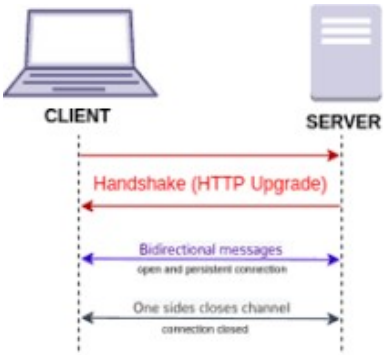
sikertelen esetben:

```
{
  "message_type": "private ",
  "content" : ”A név már foglalt, válassz másikat!",
  "timestamp": "2023-11-24T12:49:54",
  "sender": " server ",
  "receiver": " krisztian"
}
```

Kilépés:

Az üzenet tartalma @exit kell legyen, ebben az esetben a szerver törli a usert a listából és zárja a kapcsolatot.

A program dokumentációja



a WebSocket működése leegyszerűsítve

A chat projectem szerver részét Python-ben írtam, a kliens részét a tesztelés miatt először Python-ben, majd ezt továbbfejlesztve írtam a végleges Java verziót, amely grafikus felülettel is rendelkezik. A kommunikáció során WebSocket (RFC 6455)-eket használtam, amelyben JSON formában küldődnek az üzenetek. A websocket azért jó a chat megvalósításához, mert egytelen kialakított TCP kapcsolattal kétirányú kommunikációs csatornákat biztosít két eszköz között. A Python klienst nem fejlesztettem tovább a szerverrel együtt, így nem igazán működik, de csatoltam azt is.

A szerver:

Többszálú, OOP paradigmát követő Python programot igyekeztem készíteni. Az osztályok:

- **Message** (típus, tartalom, időpont, küldő, címzett)
  - to_json(self): visszaadja a példányból készített JSON objektumot, hogy lehessen küldeni
  - from_json: osztálymetódus, a kapott json objektből Message típusú példányt állít elő.

Azért nem végig JSON-nel dolgozom, mert a függvényekben könnyebb feldolgozni az üzenetet osztálypéldányként, mint JSON objektumként.

- **User** (név, conn)
  - send_message(message: Message): a kapott message objektumot JSON-be átalakítva elküldi a kialakított (megkapott conn referencia) kapcsolaton keresztül. Itt zajlik a tényleges hálózati üzenetküldés. A Router osztály függvényei szelektíven hívják meg az egyes felhasználók ezen metódusát.

- **MessageRouter:**

Egy listában tárolja az aktív felhasználókat, fel- és lecsatlakozás esetén frissíti a listát. Mindkét esetben küld egy broadcast-ot az eseményről.

A posta szerepét tölti be, lényegében minden felcsatlakozó user esetén létrejön egy külön szál (handle_client), és ezek megkapják az egyetlen router példány referenciáját. (és persze a klienssel kialakított WebSocket csatorna referenciáját)

Ezenkívül MySQL segítségével a szerver eltárolja a broadcast üzeneteket és felcsatlakozás, autentikálás után elküldi a régebbi üzeneteket a kliensnek.

Itt vannak kifejtve a privát- és publikus üzenetek metódusai. A **privát üzenetnél** például bejárja a felhasználók listáját, és ha megtalálja a címzettet, akkor elküldi az üzenetet a példány send_message metódusával. Ha nem sikerült kézbesíteni az üzenetet, visszaküld egy privát üzenetet a küldőnek erről. Publikus üzenet esetén mindenkinek elküldi az üzenetet.

A fő metódusok:

- ***start_server(ip, port):***
  Itt jön létre a message router példány és a szerver socket-je. Egy végtelen ciklusban figyeli és fogadja az új kliens kapcsolatokat, ezt eltárolja egy kliens socket-be. Minden új kliens kapcsolatnak indít egy külön szálat (handle_client metódus), átadva a socket-et és a router referenciáját.

- **handle_client:** Ez fut minden kliens thread-én, folyamatosan figyeli a kliens kapcsolaton keresztül érkező üzeneteket, ha egy null érkezik, leáll a ciklus és a szál. (bontódik a kapcsolat)
  Ha „user_info" típusú üzenet érkezett, akkor regisztrálja a kliensnek a választott nevet. Amennyiben foglalt a név, egy random 3 számjegyű számot hozzátesz.
  Ezenkívül megvizsgálja az érkezett üzenetet, a parancsokat lekezeli (pl @users-re viszsaküldi a felhasználók listáját, @exit-re lecsatlakoztatja a klienst. @newName-mel új nevet állít be a felhasználónak, ha az nem foglalt, és broadcastben értesíti a felhasználókat erről.

Mentés, beolvasás adatbázissal

Az SQL kapcsolatot a routerben hozom létre. A lekérdezések cursor segítségével történnek. A mysql connector csomagot használom (kell telepíteni). Csak a publikus üzeneteket mentem el az adatbázisba. Ezeket lekérdezem és eltárolom egy Message listába, ezt a listát végigjárva elküldöm mindenkinek belépéskor a régi üzeneteket.

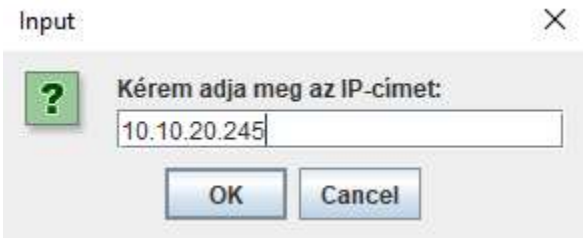
- Mivel egyelőre nem használok jelszót a belépéshez, a privát üzeneteket nem tárolom. Esetleg a kliens részben tennék egy opciót, hogy el lehessen menteni az üzeneteket egy TXT-be, ha kell)
- Azt, hogy mennyi időre visszamenőleg töltse/tárolja az üzeneteket, még nem gondoltam ki. Esetleg írnék egy event-et a db-ben, amely bizonyos időszakonként vagy bizonyos mennyiség után törli az üzeneteket a táblából.
- Ha nem sikerül kialakítani az adatbázissal a kapcsolatot, attól még tud menni a szerver. A táblát manuálisan hoztam létre, az sql utasításai mellékelve vannak.

id	type	content	sender	receiver	time
16	public	Sziasztok	Krisztián	all	2023-11-24 14:13:20
17	public	hello	Krisztián	all	2023-11-24 14:13:22
18	public	sziasztok	Béla	all	2023-11-24 14:16:56

A kliens:

igyekeztem egy felhasználóbarát, platformfüggetlen asztali alkalmazást készíteni egy általam jól ismert OOP nyelven, ezért a Java-t használtam és annak awt komponensgyűjteményét használtam. A futtatható állomány a chat.jar, amelyhez szükséges a java fordító megléte.

Mivel a kliens nem tud semmi információt a szerverről, csak a kommunikációs protokollt (websocket csatorna kialakítása valamilyen paraméterekkel, JSON üzenetek felépítése, azok küldése és fogadása), indításkor meg kell adni a dialógusban a kiszolgáló IP-címét és azt a portot, amelyen keresztül az fogadja a websocket csomagokat. A bekérés addig zajlik, amíg nem siekrül kapcsolódni, vagy be nem zárjuk a párbeszédablakot.



Igyekeztem lekezelni minden pontot, ahol a program leáll, mert előtte le kell zárni a webSocket-et is. A @help paranccsal kiírja az elérhető parancsokat: pl. @newName, @private (szintaktika), @restart, @exit, @users…

Kapcsolódás után indul egy mellékszál, amely végtelen ciklusban figyeli az érkező üzeneteket.

A jobboldali user lista mindig frissül automatikusan, ha valaki belép/kilép/nevet vált, de a gombbal is lehet kérni. A **privát** üzenetnél a szintaktika*: @private név -> üzenet*. Ha rákattintunk valaki nevére, akkor kiegészíti nekünk a program, így csak az üzenetet kell odaírni. A privát üzenetek is ugyanazon a falon jelennek meg, de csak az adott user kliensén. Enterrel is lehet küldeni az üzenetet. Üres üzenetet nem enged küldeni. Ha a kapcsolattal baj lesz (lezárja a szerver, hiba keletkezik, stb.), akkor megkérdezi a program, hogy újrainduljon-e.